Computer Graphics – 236216

Winter 2024/2025

HW4 – Advanced Features

Release Date (Week 9): 12/1/2025

Submission Date: 5/2/2025

Description:

In this exercise, you are required to upgrade your renderer from the previous assignments to include some advanced features from the following list. The features differ in their difficulties, so in order to normalize the difficulty, each feature is assigned points according to its difficulty. You are required to implement as many features as you like, as long as you gain at least 9 points.

You are free to design a reasonable user interface for each feature you decide to implement.

The features list:

- **Shadow volumes** (4.5 Points):

  Add shadow effect for the whole scene using the shadow volumes algorithm.

- **Shadow maps** (4 points):

  Add shadow effect for the whole scene using the shadow maps algorithm.

- **3D volumetric texture** (4 Points):

  For this feature, you need to implement volumetric procedural texture mapping as discussed in class. You can use the article mentioned in class. *This option has the minimal requirements of texture mapping for wood and marble, and any other one interesting texture you 'invent'.*

- **2D parametric texture** (2.5 Points)

  For this feature, you are expected to read texture map defined using a raster image and use the UV values of the vertices of the polygons to derive intensity for each pixel during the scan conversion process. In order to determine the UV for each vertex of polygon (only if it exists, if the input itd file contains surfaces), use the following code:

```
float *UV;
if ((UV = AttrGetUVAttrib(PVertex -> Attr, "uvvals")) != NULL)
{

/* UV values are in UV[0] and UV[1]. */

}
```

  Note the UV values can be in arbitrary range and are not necessarily in [0, 1]^2. Go over all polygons in an object and based on the min/max UV values detected, normalize, and scale the image so it will cover the entire surface.

- **Bump Mapping** (2.5 Points)

  Implement a way to modulate the normal of the shading equations, either using a parametric texture image that defines the normal perturbation (and UV values at the vertices) or using a prescribed volumetric texture function for normals.

- **Transparency** (3 Points)

  Here you should implement a variation of the Z-buffer where you need to store a sorted list of 'hits' for each pixel and compute the pixels color by a blending of the colors according to their z value (in camera space). You should provide an option to control the transparency of the object and read it as an attribute of the object. The transparency attribute in Irit format can be found as "transp" x attribute where x is between 0 (opaque) and 1 (fully transparent). You can retrieve it using the following code in the parser (in *CGSkelStoreData* function):

```
if (CGSkelGetObjectTransp(PObj, &Transp))
{
        /* Transparency handling code */
}
```

  Here you could also decide to use texture maps to affect the translucency of local pixels.

- **Anti Aliasing** (2 Points)

  Here a super sampling technique will be implemented for a high-quality rendering. You will Implement the following filters:

  - box, triangle, Gaussian and sinc,
    with two kernel sizes, 3X3 and 5X5, for each filter type.

- **Animation** (2.5), **with Bezier curves** (additional 2 Points)

  Add an animation tool to your renderer. Each mouse event (sampled on Mouse release) should define a key-frame transformation for the animation. The final animation should generate a continuous piecewise linear interpolation between each two keyframes (interpolating the transformation):

  - Add an option to start and stop recording the transformations that are done to the object as key-frames (on Mouse release).
  - Add an option for playing the animation in wireframe mode (Or solid if your solid renderer is fast enough).
  - Add an option to control the speed of the animation.
  - Add an option to save the animation as a sequence of images (generated by the renderer in full solid mode) that can be combined together to generate a video of the animation.

  For extra points you can add an option to approximate the transformations as a Bezier curve, each transformation defines a control point in transformation space, yielding smooth motion. Apply this Bezier approximate for, at least, the translation transformations.

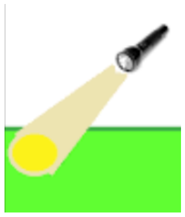
- **Motion blur** (2.5 Points)

  Add an option to support a motion blur effect of one or several moving objects in the scene. Similar to the Animation feature, but in this feature the final output is one image (instead of several) that is an integral of all the interpolation frames over time.
  - Use fading history operator to combine the frames over time. The frames should be combined in the final result image as following:
    - For the first frame:
      result_image = first frame.
    - For the next frames,
      result_image = t*result_image + (1-t)*new_frame.
      Where t is between 0 and 1. (t=0 indicates no blur).
  - Add an option to control the t value (Default value is 0.25).

- **Fog effect** (2 Points)

  Add a fog effect to the scene, by interpolating the color of the object with a fog color as a function of the depth. Add an option to choose the fog color.

- **Spotlight** (2 Points)

  Add additional type of lights: spotlights. A spotlight is a point light that lights only in some cone that is defined by a direction and an angle. Add an option to specify the direction and light angle for each spotlight in addition to the basic light source attributes.

  A Hint: while it can be implemented using shadow (light) maps, here the geometry is a cone so it can be computed in a much simpler way analytically.

Final Notes:

- If you would like to extend your renderer in other/different direction, you are more than welcome to talk to us.
- The grading of this lab is less defined compared to previous labs and the level of difficulty of the problem (and solution) will be taken into consideration.
- Submit a zip file containing your code at the web site. Do not submit the Release or Debug directories or the CGWork.ncb file. If your zip file takes more than 10 megabytes, you are doing something wrong.
- DO NOT USE any external code without permission and neither should you use other IRIT functions without permission.
- Submit electronically a single zip file, named <ID1>{_<ID2>}_HW4.zip, where ID1 and ID2 are your id numbers. The file should contain the following:
  - A readme.txt file which includes your names, ids and emails.
  - The whole homework project with the solution files (VS2019), don't include the intermediate and compilation files in the Release and Debug folders.
- Late submission should be coordinated with the TA. A penalty of 3 points will apply for each day of delay, if not justified.
- As stated at the beginning of the semester, we will be holding a "Best Rendering" competition (3 points for 1st place, 2 points for 2nd, 1 point for 3rd, in the final grade). Please also submit up to three images to the competition - add them to the zip file as <ID1>{_<ID2>}_competitionN.png, where N is 1, 2, or 3.

Good luck and enjoy!